

LOCAL RELATIVE LAYOUT OF NODE-LINK STRUCTURES

IN SPACE WITH NEGATIVE CURVATURE

Field of the Invention

The invention relates to laying out a node-link structure in a space with negative curvature such as hyperbolic space.

Background and Summary of the Invention

Lamping, J. and Rao, R., "The Hyperbolic Browser: A Focus+Context Technique for Visualizing Large Hierarchies", *Journal of Visual Languages and Computing*, Vol. 7, 1996, pp. 33-55, disclose techniques for laying out a hierarchy on a hyperbolic plane such that distance between parent and child and between siblings is approximately the same everywhere. A recursive algorithm lays out each node based on local information, allocating a wedge of the hyperbolic plane for the node's descendants. The algorithm places all the children along an arc in the wedge, at an equal distance from the parent node, and at least a minimum distance apart from each other. The layout of a node depends only on the layout of its parent and on the node structure of two or three generations starting from the parent. Therefore, the layout can be done incrementally, such as by initially laying out the nodes nearest the root and by then adding more nodes as more of the structure is traversed. Lamping et al.,

1 US-A-5,590,250, disclose similar layout techniques in which each node has a
2 data structure that includes its position and radius and, if it has children, a link to
3 a list of children; complex numbers are used to represent positions in the
4 hyperbolic plane.

5 Munzner, T., and Burchard, P., "Visualizing the Structure of the World
6 Wide Web in 3D Hyperbolic Space", in *Proceedings of the VRML '95*
7 *Symposium (San Diego, CA, December 13-16, 1995)*, ACM SIGGRAPH, 1995,
8 pages 33-38, disclose techniques for layout of a graph in the hyperbolic plane
9 that can be used with directed graphs with cycles. The hyperbolic length of an
10 edge connecting two nodes is obtained using hyperbolic functions and the
11 angles between edges incident on the two nodes. Directed graphs with cycles
12 can be embedded in hyperbolic manifolds-spaces which can wrap around and
13 close up on themselves, or in standard hyperbolic space by filling in backlink
14 edges.

15 Matsuura, T., Taniguchi, K., Masuda, S., and Nakamura, T., "A Graph
16 Editor for Large Trees with Browsing and Zooming Capabilities", *Systems and*
17 *Computers in Japan*, Vol. 24, No. 8, 1993, pp. 35-46, disclose a library program
18 which allows direct manipulation of a large tree on a display. When a node or
19 subtree is added or deleted, the data structure is updated; the layout for each
20 node is calculated based on the graph layout algorithm, and each node and
21 edge is redrawn. The layout algorithm determines, for each non-root node, a
22 relative position value that is the difference between the node's x-coordinate and

1 its parent's x-coordinate. The y-coordinate, on the other hand is determined by
2 the level of the node, and therefore is determined when the coordinates of the
3 root node are specified. When a part of the tree is deleted or a subtree is
4 added, ancestors are traced to update the layout.

5 The invention addresses problems in laying out node-link structures in a
6 space with negative curvature, such as hyperbolic space.

7 With the techniques described by Lamping and Rao and with other
8 conventional techniques for negative curvature layout, if a structure changes
9 after it is displayed, a new layout must typically be done for at least a large part
10 of the structure as changed and then it must be redisplayed. Performing layout
11 on a large structure is slow, and, when taken together with redisplay, may
12 prevent a user from effectively interacting with the changed structure. Although
13 these problems are especially acute for changing structures, they also arise
14 when operations are performed that require frequent layout of a static structure
15 or of parts of a static structure.

16 The layout techniques for static structures described by Lamping and Rao
17 have certain characteristics that mandate layout of at least a large part of the
18 structure. The layout techniques for static structures make decisions at each
19 node that depend on layout decisions higher in the tree and at its sibling nodes;
20 as a result, adding or deleting a child of a node mandates that the layout be
21 redone for all descendants of the node and of any of its siblings. Also, the
22 layout techniques for static structures save only the node's position in the

1 hyperbolic plane; any change in the structure that would change a node's
2 position requires repeating the layout of its siblings and all of its descendants.

3 The invention alleviates problems resulting from layout of large structures
4 by providing techniques that make it possible to perform local layout of a node-
5 link structure in a space with negative curvature such as hyperbolic space or the
6 hyperbolic plane. The techniques obtain nearby relationship data for an
7 element, indicating information about nearby node-link relationships. Then the
8 techniques use the nearby relationship data to obtain layout data indicating the
9 element's position relative to a parent in the space with negative curvature.

10 If the element and the parent are nodes, for example, the layout data can
11 include position displacement data indicating a distance between the parent's
12 position and the element's position and angle displacement data indicating an
13 angular difference between an incoming link to the parent and the outgoing link
14 from the parent to the element. In an especially elegant implementation, the
15 layout data include only the position displacement data and the angle
16 displacement data.

17 The nearby relationship data can be obtained by obtaining, for each of a
18 set of children of the parent, a count of grandchildren of the parent. The set of
19 children of the parent includes the element and can also include other children
20 that are being or have been laid out in the negatively curved space. The counts
21 of grandchildren can be used to obtain a radius and an angle for each of the set
22 of children. The radii and angles can then be used to obtain a position

1 displacement and an angle displacement between the parent and the element.
2 The angle displacement can be compared with a previous angle displacement to
3 determine whether to lay out children of the element.

4 The nearby node-link relationships can include only relationships among
5 the parent and the parent's children and grandchildren.

6 The new techniques can be implemented iteratively, with each iteration
7 identifying elements to be laid out and, for each identified element, obtaining
8 nearby relationship data and layout data. For example, if a series of iterations is
9 performed in response to an insertion or deletion event, the identified elements
10 can include elements affected by the insertion or deletion. The identified
11 elements for each iteration can also include elements added to the structure
12 during a preceding iteration. Before the series, a weight can be obtained for
13 each iteration, for use in obtaining layout data during the iteration.

14 The new techniques are advantageous in laying out a dynamic node-link
15 structure because when an element is deleted or inserted only a few nearby
16 elements need to be laid out again. The techniques also obtain an element's
17 position relative to a parent rather than an absolute position, making it possible
18 to change the position of an element and all its descendants by making a single
19 change or a small constant number of localized changes in the data structure.

20 The new techniques are also advantageous because they can be used in
21 a variety of situations in which it is desirable to lay out part of a node-link

1 structure. Thus, the new techniques are not only applicable to a dynamic node-
2 link structure, but also to a static structure that can only be laid out and
3 displayed in fragments because not all of the structure is available in memory.
4 The techniques are especially advantageous for laying out a tree that is a partial
5 representation of a directed graph in which nodes have multiple in-links—a
6 shared branch of such a tree need only be completely laid out once for all of its
7 occurrences, since only the relative position of the uppermost elements will differ
8 between occurrences.

9 The techniques are also advantageous for animation of a change
10 involving only part of a structure. For example, if insertions or deletions are
11 made in a structure, it may be desirable to animate the transition in a way that
12 only changes positions of elements that are near the insertion or deletion. The
13 techniques make it possible to rapidly perform a series of layouts, one for each
14 animation step, in which only the positions of elements near insertions or
15 deletions are changed. As a result, animation performance improves and a
16 simpler animation algorithm can be used. The algorithm can continuously
17 change only a small number of variables, such as an angle and radius for each
18 node that is near an insertion or a deletion.

19 Another advantage is that the new techniques provide layouts from which
20 a display can be generated starting at any arbitrary element, rather than always
21 starting at the root node or at the bottom leaves of a structure as in conventional
22 techniques.

Yet another advantage is that an element's position relative to its parent can always be expressed with adequate precision. In contrast, if an element's absolute position in a hyperbolic plane is used, a large structure could exhaust the available floating point numbers.

5 The following description, the drawings, and the claims further set forth
6 these and other aspects, objects, features, and advantages of the invention.

7 Brief Description of the Drawings

Fig. 1 is a schematic flow diagram showing how local relative layout can be performed for an element in a node-link structure.

10 Fig. 2 is a flow chart showing general acts in performing local relative
11 layout as illustrated in Fig. 1.

Fig. 3 is a schematic diagram showing general components of a machine that performs relative local layout as illustrated in Fig. 1.

14 Fig. 4 is a schematic diagram of a system.

Fig. 5 is a flow chart showing how the system of Fig. 4 can respond to events by presenting representations of a directed graph.

17 Fig. 6 is a flow chart showing how initial layout can be performed in Fig. 5.

18 Fig. 7 is a flow chart showing how layout of a changed node-link structure
19 can be performed in Fig. 5.

1 **Detailed Description of the Invention**

2 A. Conceptual Framework

3 The following conceptual framework, when taken with the conceptual
4 frameworks set forth in U.S. Patents 5,590,250 and 5,619,632, incorporated
5 herein by reference, is helpful in understanding the broad scope of the invention,
6 and the terms defined below have the indicated meanings throughout this
7 application, including the claims.

8 A "node-link structure" is a structure that includes items that can be
9 distinguished into nodes and links, with each link relating two or more of the
10 nodes. A "graph" is a node-link structure in which each link relates two nodes. A
11 "directed graph" is a graph in which each link indicates direction between the
12 nodes it relates, with one node being a source or "from-node" of the link and the
13 other being a destination or "to-node" of the link. An "acyclic directed graph" is a
14 directed graph in which the links, when followed in their indicated directions, do
15 not provide a path from any node back to itself. A "tree" is an acyclic directed
16 graph with exactly one root node such that, for any non-root node in the tree, the
17 links, when followed in their indicated directions, provide only one path that
18 begins at the root node and leads to the non-root node.

19 The "elements" of a node-link structure are its nodes and links.

1 In a node-link structure, a "node-link relationship" is a relationship
2 between elements based on paths of nodes and links between or among the
3 elements.

4 In many cases, node-link relationships can be summarized by category.
5 In a directed graph, for example, the "children" of an element are the elements
6 that can be reached from the element by following no more than one link in its
7 indicated direction. Similarly, the "parents" of an element are the elements that
8 can be reached from the element by following no more than one link opposite its
9 indicated direction. Children and parents of a node thus include links and
10 nodes, while children and parents of a link include only nodes. The
11 "descendants" of an element include all of its children, the children of its children
12 ("grandchildren"), etc. The "ancestors" of an element include all of its parents,
13 the parents of its parents ("grandparents"), etc. The "siblings" of an element
14 include all the other children of its parents. The "co-parents" of an element
15 include all the other parents of its children.

16 More generally, it is useful to distinguish node-link relationships based on
17 shortest path lengths between elements, where path length is measured by the
18 number of elements in the path. In a directed graph, for example, a node's
19 "nearest" node-link relationships would be with its incoming and outgoing links;
20 then with other nodes that are its parent and children nodes; then with other
21 incoming and outgoing links of its parent and children nodes; then with other
22 nodes that are its grandparents, siblings, grandchildren, and co-parents; and so

1 forth. A set of "nearby" node-link relationships of an element is a set of
2 relationships in which all the shortest path lengths between elements meet an
3 appropriate criterion for nearness, such as that all are shorter than a relatively
4 short maximum length such as 2, 3, or 4. An example of a useful group of
5 nearby node-link relationships is the group of relationships among an element's
6 parent, the parent's children (including the element), and the parent's
7 grandchildren.

8 An item of data "identifies" an element in a node-link structure if the item
9 of data provides sufficient information to distinguish the element from other
10 elements in the structure. For example, an item of data identifies an element if
11 the item of data can be used to access data relating to the element rather than to
12 access data relating to other elements.

13 The terms "space" and "position" have related meanings as follows: A
14 "space" is a set of "positions" over any pair of which a distance measure for the
15 space can be applied to obtain a distance between the pair of positions.
16 Examples of types of spaces include one-dimensional spaces such as lines or
17 rays; other n-dimensional spaces; continuous spaces; discrete approximations of
18 continuous spaces; and so forth.

19 A "planar unit disk" or "unit disk" is a two-dimensional Euclidean space
20 bounded by a circular perimeter, with first and second perpendicular axes that
21 cross at the center of the perimeter, and with a radius along each axis from the

1 center to the perimeter of one. As a result, each position in the unit disk can be
2 uniquely identified by two coordinates, each between +1 and -1.

3 A "space with negative curvature" is a space in which parallel lines
4 diverge. Therefore, through any position in a space with negative curvature that
5 is not on a given straight line, there are multiple other straight lines parallel to
6 the given straight line. An example of a space with negative curvature is
7 hyperbolic n-space. A "hyperbolic plane" is a hyperbolic 2-space.

8 An operation "lays out" a structure in a space if the operation obtains data
9 indicating positions in the space for elements of the structure. The structure is
10 "laid out" in the space if data indicating such positions in the space has been
11 obtained. The data could indicate absolute positions, such as with coordinate
12 values measuring displacements from a set of universal references such as a
13 coordinate origin, or the data could indicate relative positions, such as with
14 coordinate values measuring displacements from a set of references based on
15 the position of another element.

16 Data indicate an element's "position relative to a parent" if the data
17 indicate one or more displacements that can be used to obtain an absolute
18 position of the element from the absolute position of the parent. For example, a
19 set of displacements that define a vector from a parent's position to an element's
20 position indicate the element's position relative to the parent. Examples of types
21 of displacements include position displacements indicating distances between
22 positions, whether along coordinate axes or in the form of a magnitude, and

1 angle displacements indicating angle differences, such as between an incoming
2 link to a parent and an outgoing link from the parent to the element.

3 An operation "applies a criterion" if the operation uses a criterion to reach
4 a determination, such as whether to lay out children of an element.

5 As used herein, a "series of iterations" is a series of operations that can
6 be divided into two or more consecutive parts, each of which is referred to herein
7 as an iteration. Although iterations in a series may differ significantly, each
8 iteration after the first can typically use starting data produced by the preceding
9 iteration to obtain ending data. Typically, each iteration's ending data can in
10 turn be used by the following iteration as its starting data.

11 The term "navigation signal" is used herein to mean a signal that indicates
12 that the user has greater interest in a part of a node-link structure than in other
13 parts. For example, an "expand signal" indicates a request to present a
14 representation of a graph in which the representation of an element of the graph
15 is expanded, while a "contract signal" indicates a request to present a
16 representation of a graph in which the representation of an element of the graph
17 is contracted. Other examples include requests to present a part of the node-
18 link structure at a specific position, which can be done by selecting a bookmark
19 or the like or by a point and click operation requesting that a feature pointed to
20 be moved to a center of focus.

1 A signal "requests a change" in a node-link structure if the signal requests
2 a change in one or more elements of the structure, such as an insertion or
3 deletion of one or more elements or an operation such as moving or copying that
4 can be implemented by a combination of insertions and deletions.

5 A "processor" is a component of circuitry that responds to input signals by
6 performing processing operations on data and by providing output signals. The
7 input signals may, for example, include instructions, although not all processors
8 receive instructions. The input signals to a processor may include input data for
9 the processor's operations. The output signals similarly may include output data
10 resulting from the processor's operations.

11 A "network" is a combination of circuitry through which a connection for
12 transfer of data can be established between machines.

13 B. General Features

14 Figs. 1-3 show general features of the invention.

15 Fig. 1 illustrates how an element in node-link structure 10 can be laid out
16 in a space with negative curvature. As illustrated by the dashed line to box 20,
17 an element in node-link structure 10, illustratively node 22, has several nearby
18 node-link relationships. As shown in box 20, nearby node-link relationships can,
19 for example, include relationships resulting from a link to node 22 from parent
20 node 24, links from parent node 24 to sibling nodes 26 through 28, and links
21 from node 22 to children nodes 30 through 32.

1 As illustrated by the solid line from node-link structure 10 and by the
2 dashed line from box 20, nearby relationship data 40 is obtained, indicating
3 information about nearby node-link relationships of node 22. Then, layout data
4 42 for node 22 can be obtained based on nearby relationship data 40. Other
5 information about node-link structure 10 could also be used in obtaining layout
6 data 42, but such other information should preferably not include information
7 about distant node-link relationships, to ensure that node 22 only needs to be
8 laid out again when nearby node-link relationships are modified.

9 Layout data 42 may indicate various information about node 22 and its
10 position in a space with negative curvature, including at least a position of node
11 22 relative to parent node 24. For example, as shown in box 44, layout data 42
12 can indicate a vector D indicating the distance and direction in the negatively
13 curved space from parent node 24 to node 22. Just as position of a child can be
14 defined relative to a parent's position, the direction from parent to child can be
15 defined relative to the direction from grandparent to parent (or, if the parent is
16 the root, relative to an initial direction). The vector D can thus indicate the
17 position displacement from the parent and the angle displacement from the
18 grandparent to parent direction.

19 In Fig. 2, the act in box 100 begins by obtaining nearby relationship data
20 for an element in a node-link structure, indicating information about nearby
21 node-link relationships of the element. As indicated by the inner dashed line

1 around box 100, nearby relationship data can be obtained for each of a set of
2 elements.

3 Based on the nearby relationship data obtained in box 100, the act in box
4 102 then obtains layout data indicating an element's position relative to a parent
5 in the negatively curved space. As indicated by the inner dashed line around
6 box 102, layout data can similarly be obtained for each of a set of elements for
7 which nearby relationship data was obtained in box 100. As indicated by the
8 outer dashed line around boxes 100 and 102, nearby relationship data can be
9 obtained for another element or for another set of elements after obtaining an
10 element's layout data.

11 Machine 150 in Fig. 3 includes processor 152 connected for receiving
12 data indicating user signals from user input circuitry 154 and for providing data
13 defining images to display 156. Processor 152 is also connected for accessing
14 node-link data 158, which define at least part of a node-link structure. Processor
15 152 is also connected for receiving instruction data 160 indicating instructions
16 through instruction input circuitry 162, which can illustratively provide
17 instructions received from connections to memory 164, storage medium access
18 device 166, or network 168.

19 In executing the instructions indicated by instruction data 160, processor
20 152 obtains nearby relationship data for an element, indicating information about
21 nearby node-link relationships of the element. Processor 152 can access node-
22 link data 158 to obtain information about node-link relationships. Then, based

1 on the nearby relationship data, processor 152 obtains layout data indicating the
2 element's position relative to a parent in a space with negative curvature.

3 As noted above, Fig. 3 illustrates three possible sources from which
4 instruction input circuitry 162 could receive data indicating instructions--memory
5 164, storage medium access device 166, and network 168.

6 Memory 164 could be any conventional memory within machine 150,
7 including random access memory (RAM) or read-only memory (ROM), or could
8 be a peripheral or remote memory device of any kind. More generally, memory
9 164 could be a combination of more than one type of memory component.

10 Storage medium access device 166 could be a drive or other appropriate
11 device or circuitry for accessing storage medium 170, which could, for example,
12 be a magnetic medium such as a set of one or more tapes, diskettes, or floppy
13 disks; an optical medium such as a set of one or more CD-ROMs; or any other
14 appropriate medium for storing data. Storage medium 170 could be a part of
15 machine 150, a part of a server or other peripheral or remote memory device, or
16 a software product. In each of these cases, storage medium 170 is an article of
17 manufacture that can be used by machine 150. Data units can be positioned on
18 storage medium 170 so that storage medium access device 166 can access the
19 data units and provide them in a sequence to processor 152 through instruction
20 input circuitry 162. When provided in the sequence, the data units form
21 instruction data 160, indicating instructions as illustrated.

1 Network 168 can provide instruction data 160 received from machine 180.
2 Processor 182 in machine 180 can establish a connection with processor 152
3 over network 168 through network connection circuitry 184 and instruction input
4 circuitry 162. Either processor could initiate the connection, and the connection
5 could be established by any appropriate protocol. Then processor 182 can
6 access instruction data stored in memory 186 and transfer the instruction data
7 over network 168 to processor 152 so that processor 152 can receive instruction
8 data 160 from network 168. Instruction data 160 can then be stored in memory
9 164 or elsewhere by processor 152, and can be executed.

10 C. Implementation

11 The general features described above could be implemented in numerous
12 ways on various machines to present node-link representations. An
13 implementation described below has been implemented on a PC-based system
14 running the 32 bit versions of Microsoft Windows and executing code compiled
15 from C++ language source code.

16 C.1. System

17 In Fig. 4, system 200 includes PC processor 202, which is connected to
18 display 204 for presenting images and to keyboard 206 and mouse 208 for
19 providing signals from a user. PC processor 202 is also connected so that it can
20 access memory 210 and client 212. Memory 210 can illustratively include
21 program memory 214 and data memory 216. Client 212 is a source of

1 information about a directed graph, which could be a combination of routines
2 and data stored in memory 210 or could be independent of memory 210 as
3 shown. For example, processor 202 could communicate with client 212 through
4 a network.

5 The routines stored in program memory 214 can be grouped into several
6 functions. Grapher routines 220 create and modify a data structure representing
7 the directed graph defined by the information from client 212. Walker routines
8 222 respond to navigation signals and other user signals from keyboard 206 and
9 mouse 208 by obtaining information from the directed graph data structure.
10 Painter routines 224 provide signals to display 204 to cause it to present
11 representations of the directed graph data structure. Math routines 226 can be
12 called to obtain positions of elements of the directed graph in a layout space.

13 Data memory 216 in turn contains data structures accessed by processor
14 202 during execution of routines in program memory 214. Directed graph data
15 structure 230, as noted above, can be created and modified by grapher routines
16 220 and can also be accessed by walker routines 222 and painter routines 224.

17 Further details about the implementation of directed graph data structure
18 230 are set forth in copending coassigned U.S. Patent Applications
19 09/DDD,DDD (Attorney Docket No. D/98205Q3), entitled "Controlling Which Part
20 of Data Defining a Node-Link Structure is in Memory", and 09/EEE,EEE
21 (Attorney Docket No. D/98205Q4), entitled "Node-Link Data Defining a Graph
22 and a Tree Within the Graph", both incorporated herein by reference.

1 Node position data 232, which can be linked to or included within directed
2 graph data structure 230, can include positions of nodes in a negatively curved
3 space such as a hyperbolic plane and in a rendering space such as a two-
4 dimensional unit disk. Node position data 232 can be accessed by routines in
5 program memory 214.

6 The routines in program memory 214 can also access various
7 miscellaneous data structures 234. Data structures 234 may, for example,
8 include an extra data structure for mapping from a pair of node IDs to a link ID,
9 implemented as a standard heap; this extra data structure allows lookup and
10 insertion of a link ID in constant expected time.

11 C.2. Responding to Events

12 Fig. 5 shows how the system of Fig. 4 can respond to events by
13 presenting representations of a graph.

14 In box 300, client 212 begins by obtaining a starting graph and by loading
15 an initial set of elements into memory, such as through calls to create nodes as
16 described in copending coassigned U.S. Patent Application 09/DDD,DDD
17 (Attorney Docket No. D/98205Q3), entitled "Controlling Which Part of Data
18 Defining a Node-Link Structure is in Memory", incorporated herein by reference.
19 Expansion flags define a tree within the initial set of elements, as described in
20 copending coassigned U.S. Patent Application 09/EEE,EEE (Attorney Docket
21 No. D/98205Q4), entitled "Node-Link Data Defining a Graph and a Tree Within

1 the Graph", incorporated herein by reference. Client 212 also makes
2 appropriate calls to routines in memory 214 for layout of the tree in a hyperbolic
3 plane, for mapping the tree from the hyperbolic plane to a unit disk with the root
4 node at the disk center, for painting the mapped tree, and for presentation of the
5 painted version on display 204 by swapping a double buffer, all in box 300.

6 In box 302, client 212 receives an event relating to the graph. The event
7 could result from a navigation signal, an editing signal, or another type of signal
8 from a user. Alternatively, the event could be received from another source,
9 either within or external to system 200. In either case, the event could take the
10 form of a call from within client 212, from one of the routines in memory 214, or
11 from other instructions executed by processor 202. A series of received events
12 could be held in a queue, so that box 302 could involve popping an event from a
13 queue.

14 In response to the event received in box 302, client 212 initiates an
15 appropriate response by making one or more calls to routines in memory 214.
16 As indicated by box 304, the response depends on the type of event, so that a
17 branch is taken based on the event.

18 The event may be a non-animated event, such as an orientation shift
19 event, a stretch event, or a dragging event. An orientation event can result
20 when the user indicates a new orientation for the root node. A stretch event can
21 result when the user indicates a new stretch factor for the displayed
22 representation. A dragging event, for example, can result when the user selects

1 a position within the representation, such as by a mouse down click, and
2 requests that it be moved by an appropriate gesture or other signal.

3 Client 212 begins the response to a non-animated event in box 310 by
4 obtaining any information needed for responding to the event. For an orientation
5 event, the information obtained in box 310 can include the new orientation. For
6 a stretch event, the information obtained in box 310 can include the new stretch
7 factor.

8 For a dragging event, obtaining information in box 310 is somewhat more
9 complicated. Client 212 could obtain a node identifier (node ID) of the node
10 nearest the selected position and could also obtain information about the
11 requested motion. These items of information could be obtained in much the
12 same way as illustrated by the function find-nearest-node described at cols. 71-
13 72 and as described in relation to Fig. 14 of U.S. Patent 5,590,250, incorporated
14 herein by reference.

15 When client 212 has obtained the necessary information in box 310, it
16 can conclude with appropriate calls to walker routines 222 and painter routines
17 224 for layout, mapping, and painting. For an orientation event, the root node
18 must be laid out at the new orientation. For a stretch event or a dragging event,
19 layout is not needed. For a stretch event, the call to walker routines 222 must,
20 however, include the new stretch factor, for use in mapping. Similarly, for a
21 dragging event, the call to walker routines 222 must include the node ID of the
22 nearest node and the next position along the path of motion, for use in mapping.

1 In box 312, walker routines 222 could first perform any necessary layouts
2 in the hyperbolic plane, and could also lay out any pending edits of the tree.
3 Then, in box 314, walker routines 222 could map the tree into the unit disk,
4 beginning with a starting node at a starting position, in the manner described in
5 copending coassigned U.S. Patent Application 09/CCC,CCC (Attorney Docket
6 No. D/98205Q2), entitled "Mapping a Node-Link Structure to a Rendering Space
7 Beginning from any Node", incorporated herein by reference. For example, in
8 response to a dragging event, the starting node could be the nearest node
9 identified in box 310 and the starting position could be the next position along
10 the path of motion. The starting node and starting position previously used for
11 mapping could be used in response to an orientation or stretch event.

12 When the tree has been mapped, painter routines 224 can be called to
13 paint the mapped tree in a display buffer, in box 316. During painting, painter
14 routines 224 can mark new edits that occur in the tree as a result of node
15 creation as described in copending coassigned U.S. Patent Application
16 09/EEE,EEE (Attorney Docket No. D/98205Q4), entitled "Node-Link Data
17 Defining a Graph and a Tree Within the Graph", incorporated herein by
18 reference. Each edit can be marked by setting a flag or storing other
19 appropriate data. When painting is completed, a swap of display buffers can be
20 performed to present the tree as painted, thus providing a representation of the
21 graph.

1 As noted above, these events are currently implemented as non-animated
2 events. In response to an orientation event, the representation pivots to the new
3 orientation, typically around the node at the focus of the display region.
4 Similarly, in response to a stretch event, the representation expands or contracts
5 radially, typically around the node at the focus. In response to a dragging event,
6 the representation moves at a rate determined by the input signal. Client 212
7 could, however, provide an animated response to an orientation event, a stretch
8 event, or a dragging event by converting the requested change into an
9 equivalent sequence of smaller events, and issuing a series of calls in box 310,
10 one call for each of the smaller events.

11 Fig. 5 also illustrates responses to two different types of events which are
12 always treated as animated in the current implementation. The first type is a
13 bookmark or click event, in response to which one node's position is moved
14 during an animated sequence and other elements move to accommodate the
15 one node's movement. The second type is an insert/delete event, in response to
16 which one node remains stable during an animated sequence in which some
17 elements are contracted, others are expanded, and still others move to
18 accommodate the contractions and expansions.

19 A bookmark or click event could result when the user selects an item in a
20 menu or other collection of bookmarks or selects a position within the
21 representation with a mouse down-up click. In response to an event of this type,
22 client 212 obtains a node ID and a destination position in the unit disk. In the

1 case of a bookmark event, the node ID and destination position are previously
2 stored and can be retrieved from memory. In the case of a click event, client 212
3 could obtain a node ID of the node nearest the selected position in much the
4 same way as the function find-nearest-node described at cols. 71-72 of U.S.
5 Patent 5,590,250, incorporated herein by reference, and the destination could
6 be a default position, such as the center of the unit disk.

7 In box 320, client 212 could call walker routines 222 with the node ID and
8 destination position. Walker routines 222 can respond by performing an
9 animation loop to present a sequence of representations in which the node
10 moves from its previous position to the destination position. In box 320, walker
11 routines 222 begin by setting up a sequence of node/position pairs, each
12 including the node ID and a position in the unit disk. The positions can be
13 obtained by obtaining a total translation from the previous position to the
14 destination position, then obtaining and repeatedly composing an nth root of the
15 total translation with a current translation as described in relation to boxes 470,
16 472, and 482 in Fig. 12 of U.S. Patent 5,619,632, incorporated herein by
17 reference. The number of node/position pairs can be large enough to ensure a
18 smooth animation from the previous position to the destination position, with
19 features representing elements of the structure maintaining object constancy
20 during the animation. As an alternative to the nth root approach, the positions
21 could be obtained by selecting an appropriate number of points along an
22 appropriately chosen arc in the hyperbolic plane from the previous position to

1 the destination position. The arc could be chosen to compromise between a
2 straight line, which can appear unnatural, and the arc the node would have
3 taken in the nth root method, which can require an excessive number of
4 animation steps to appear smooth. The number of points could be chosen to
5 ensure satisfactory animation.

6 In obtaining positions in box 320, orientation can be preserved as
7 described in relation to Fig. 15 of U.S. Patent 5,590,250, incorporated herein by
8 reference. Alternatively, transformations or rotations can be chosen so that the
9 position of a particular point on the boundary of the unit circle is preserved. The
10 point chosen could, for example, be the point on the circle in the opposite
11 direction from the direction in which the children of the root were laid out.

12 Walker routines 222 then perform an iteration of the animation loop for
13 each node/position pair in the sequence, as indicated in box 322. In box 324,
14 walker routines 222 could first lay out in the hyperbolic plane any pending edits
15 of the tree, as described above in relation to box 312. Then, in box 326, walker
16 routines 222 could map the tree into the unit disk, beginning with the node and
17 position from the next node/position pair as the starting node and the starting
18 position, as described above in relation to box 314.

19 When the tree has been mapped, painter routines 224 can be called to
20 paint the mapped tree in a display buffer, in box 328. During painting, painter
21 routines 224 can mark new edits that occur in the tree as a result of node
22 creation as described above in relation to box 316. When painting is completed,

1 a swap of display buffers can be performed to present the tree as painted, thus
2 providing a representation of the graph.

3 When a new edit is marked in box 328 by painter routines 224, the new
4 edit is laid out during the next iteration, in box 324. As a result, the animated
5 sequence of representations, rather than showing a static node-link structure as
6 in U.S. Patent 5,629,632, shows a dynamic node-link structure. The edits,
7 however, serve primarily to add features representing new nodes along the outer
8 perimeter of the representation as the representation makes the transition from
9 the previous position to the destination position. As a result, the added features
10 do not interfere with or reduce the perception of object constancy for features
11 representing other elements.

12 An insert/delete event could result when the user requests expansion or
13 contraction of a node or requests some other modification of the graph or the
14 tree. An insert/delete event could also be received in the form of a call, and
15 could thus provide a mechanism for automatic modification of the graph or tree
16 without concurrent human control.

17 In response to an event of this type, client 212 can first make appropriate
18 calls to routines in memory 214 to determine whether the requested modification
19 of the graph or tree is acceptable, in box 330. For example, a technique for
20 determining whether an expand signal is acceptable is described in relation to
21 Fig. 7 of copending coassigned U.S. Patent Application 09/EEE,EEE (Attorney

1 Docket No. D/98205Q4), entitled "Node-Link Data Defining a Graph and a Tree
2 Within the Graph", incorporated herein by reference.

3 If the requested modification of the graph or tree is acceptable, client 212
4 can modify the graph or tree accordingly, making calls to routines in memory 214
5 as necessary. In the process of making the modification, each element that
6 could be inserted, deleted, or changed by the modification, referred to herein as
7 an "affected element", is marked, such as by setting a flag or storing other
8 appropriate data. If a node is selected by an expand signal or a contract signal,
9 its parent is also an affected node, because the area allocated to the parent may
10 change. For most other insert/delete events, only nodes that are inserted or
11 deleted are affected. Client 212 can then select a node as a stable node to be
12 held at its previous position during animated presentation of the modification.

13 In many cases, the starting node used previously in mapping can be
14 selected as the stable node, and it can be held at the previous starting position.
15 In other cases, it may be desirable for client 212 to select a different stable
16 node; for example, a node that is being expanded could be selected as the
17 stable node, to be held at its current position, which thus becomes the new
18 starting position. Therefore, unless client 212 selects a different stable node,
19 the previous starting node and starting position are usually retained. But where
20 the previous starting node is being deleted, another node must be selected as
21 the default stable node subject to change by client 212.

1 When a deletion is being made, walker routines 222 can be called with
2 the node IDs of the node being deleted and of its closest ancestor that will
3 remain in the tree being mapped after deletion. This ancestor can be found by
4 walking upward from the node being deleted until an ancestor is reached that is
5 not being deleted by the current insert/delete event.

6 In response to this call, walker routines 222 can test whether the node
7 being deleted is the previous starting node. If so, the identified ancestor can be
8 selected to replace it as the starting node. If the ancestor has been recently
9 mapped to a position that is displayed and that is available, that position can be
10 selected as the starting position. If the ancestor has not been recently mapped,
11 or was mapped to a position that is not displayed or that is not available because
12 another element has now been mapped there, the starting position can be the
13 center of the unit disk.

14 Also in box 330, client 212 could call walker routines 222 with the stable
15 node ID and position. Walker routines 222 can respond by performing an
16 animation loop to present a sequence of representations in which, first, deleted
17 nodes are contracted at their previous positions, and then inserted nodes are
18 expanded at their new positions, all while the stable node is held at its previous
19 position. If the stable node cannot be held at its previous position because it
20 was not recently mapped or was mapped to a position that is not displayed or is
21 not available, it can be shifted to that position after deleted nodes are contracted
22 with the previous starting node at the previous starting position, resulting in a

1 sudden movement between contraction and expansion. Walker routines 222
2 begin by setting up a sequence of weights to govern the rate at which the area
3 allocated to each affected node changes during contraction and expansion. The
4 weights are separated by sufficiently small increments to preserve object
5 constancy during animation.

6 Walker routines 222 then perform an iteration of the animation loop for
7 each weight in the sequence, as indicated in box 332. In box 334, walker
8 routines 222 could first lay out in the hyperbolic plane the affected nodes and
9 any pending edits of the tree, using the iteration's weight. Then, in box 336,
10 walker routines 222 could map the tree into the unit disk, beginning with the
11 stable node and position, as described above in relation to box 314.

12 When the tree has been mapped, painter routines 224 can be called in
13 box 338 to paint the mapped tree in a display buffer. During painting, painter
14 routines 224 can mark new edits that occur in the tree as a result of node
15 creation as described above in relation to boxes 316 and 328. When painting is
16 completed, a swap of display buffers can be performed to present the tree as
17 painted, thus providing a representation of the graph.

18 Whether there are pending edits or not, a series of iterations of the
19 animation loop beginning in box 332 produces representations of a dynamic
20 node-link structure because of the deletions and/or insertions. In addition,
21 affected elements move to new positions from their positions prior to the

1 deletions and insertions. The technique has been successfully implemented to
2 produce object constancy during these movements.

3 After a representation is provided in box 316 or after an animation
4 sequence is completed in box 322 or 332, another event can be received in box
5 302, as indicated by the circles labeled "A" in Fig. 5.

6 Animation details relating to the loops that begin in boxes 322 and 332
7 are discussed in more detail in copending coassigned U.S. Patent Application
8 09/AAA,AAA (Attorney Docket No. D/98205), entitled "Presenting Node-Link
9 Structures with Modification", incorporated herein by reference.

10 C.3. Layout

11 Fig. 6 shows how layout can be initially performed in box 300 in Fig. 5.
12 Fig. 7 shows how layout of a changed node-link structure can be performed in
13 boxes 312, 324, and 334.

14 As shown in box 350, walker routines 222 begin initial layout by obtaining
15 the root node ID and using it to access data relating to the root node in directed
16 graph data structure 232. In box 352, walker routines 222 lay out the root node
17 by making a call to math routines 226 with an angle width. This could be any
18 suitable angle that produces a desirable result. The angles 2π and $\pi/2$ have
19 been successfully used, with 2π appropriate for a center layout style and with
20 $\pi/2$ appropriate for a top, bottom, right, or left layout style. An interface could
21 also be provided to modify this angle to obtain desirable results.

1 In response, math routines 226 lay out the root node at the origin of the
2 unit circle, at coordinates (0, 0); with an upward orientation, at coordinates (0, 1);
3 and with an angle half the angle width. Then, walker routines 222 push the root
4 node ID onto the front of a queue in box 354.

5 In the remainder of Fig. 6, walker routines 222 iteratively traverse a set of
6 elements of the tree defined by directed graph data structure 232, until the
7 queue is empty, as indicated in box 360. Each iteration begins by getting the
8 node ID from the back of the queue and using it to access data relating to the
9 identified node in directed graph data structure 232.

10 In each iteration, the test in box 370 determines whether the node has
11 already been walked in this traversal. If not, in box 372, walker routines 222
12 mark the node walked; get the node IDs of the children; identify which of the
13 children are nonexpanded leaves, i.e. leaf nodes with no expanded incoming
14 links; obtain the number of the children that are in the visible tree, as explained
15 below; and call math routines 226 with the number N of children of the node that
16 are in the visible tree to obtain arrays of angles and radii for the children in the
17 visible tree, also explained below.

18 For the operations in box 372, the number of children in the visible tree N,
19 which is a type of nearby relationship data, can be calculated in either of two
20 ways: If the current traversal is part of a sequence of steps that add nodes, N is
21 equal to the sum of the number of children prior to the traversal plus the number
22 of children being added. If the current traversal is part of a sequence of steps

1 that remove nodes, N is simply equal to the number of children prior to the
2 traversal.

3 The arrays of angles and radii can be obtained in box 372 in a variety of
4 ways. In one successful implementation, each radius is set to the value 0.7,
5 while each angle is set to the smaller of $((N * \pi)/18)$ and π . Thus, for $N < 18$, a
6 node's angle will depend on the number of its children that are in the visible tree.

7 Then, in box 374, walker routines 222 call math routines 226 to lay out the
8 children.

9 Two general principles of layout are applied in implementing box 374:
10 First, spacing and angle between nodes are determined based only on
11 information about nearby elements in the tree, i.e. nearby relationship data; and,
12 second, the layout information obtained for each node indicates the relative
13 position of a node to its parent in such a way that the position of a node and all
14 its children can be shifted by a small change in the data structure.

15 A general strategy that can be followed is to start with a child's radius and
16 angle from box 372, obtain approximate distances the child needs, use the
17 approximate distances to obtain a distance from the parent, then use the
18 distance from the parent to obtain more precise distances for the child, and then
19 optionally use the more precise distances to obtain even more precise
20 distances, and so forth.

1 According to the general strategy, if a child has radius R and angle Θ
2 from box 372, the approximate distances $D1$ and $D2$ can be calculated as
3 $\sinh(R)$ and $\tan(\Theta/4)$, respectively. $D1$ and $D2$ can be used to obtain a total
4 distance DT for all the children, where each adjacent pair of children are
5 separated by the greater of the sums of their $D1$ s and $D2$ s, i.e. $DT = \Sigma(\max(D1(i)$
6 $+ D1(i+1), D2(i) + D2(i+1))) + \max(D1(1), D2(1)) + \max(D1(N), D2(N))$, where the
7 summation runs from $i=1$ to $N-1$, with N the number of children for which layout is
8 being performed. If the parent has an available angle ω , the distance DP from
9 the parent can then be calculated as $\text{asinh}(DT/\omega)$. The children can then be
10 positioned along the circumference of a circle of radius DP centered at the
11 parent with the angles between them proportional to their separations.

12 DP can then be used to obtain more accurate distances for the children,
13 as follows:

14 $D1' = \sinh(DP) \text{asin}(\sinh(R)/\sinh(DP));$

15 $D2' = 2\sinh(DP) \text{atan}(\tan(\Theta/4)/e^{DP}).$

16 $D1'$ and $D2'$ can then be used to obtain a more accurate distance DP' to the
17 parent, as above, and so forth until a desired level of precision is reached. At
18 that point, the orientation of each child can be calculated as an angle offset from
19 the orientation of the parent.

1 Note that the distances described above are expressed in true metrics in
2 the hyperbolic plane. A distance D in the hyperbolic plane corresponds to a
3 vector in the unit circle starting at the origin and going a distance $\tanh(D/2)$.

4 The general strategy thus obtains layout information based only on
5 nearby relationship information about a node, its parent, and its siblings,
6 including information about a sibling's children in the visible tree, as described
7 above in relation to box 372. The general strategy obtains layout information
8 indicating the distance from a child to its parent and an angle representing the
9 difference in orientation between them.

10 The general strategy has been implemented in software that goes through
11 the children with two iterative loops, but uses the first distance obtained without
12 attempting to obtain more precise distances in the manner described above in
13 relation to the general strategy. The first loop obtains and temporarily saves
14 separations between adjacent children and a "slice size" for each child, and also
15 obtains a total separation. This information is then used to obtain the distance
16 to the parent. The second loop then obtains and saves the relative orientation
17 and area of each child.

18 In the software implementation, if a child has radius R and angle Θ from
19 box 372, distances $D1$ and $D2$ are calculated as set forth above in relation to the
20 general strategy. $D1$ and $D2$ for each child are added to $D1$ and $D2$ for the
21 previous child to obtain $S1$ and $S2$. A total separation ST is increased by the

1 maximum of the child's S1 and S2, except for the first and last children, for which
2 ST is increased by the maximum of the child's D1 and D2.

3 If the child's S1 is greater than its S2, S1 is saved as the child's
4 separation, D1 is initially saved as the size of the child's slice, and, for the
5 second and subsequent children, the previous child's slice size is adjusted to be
6 the minimum of its previous slice size and its S1. Conversely, if the child's S1 is
7 not greater than its S2, S2 is saved as the child's separation, D2 is initially saved
8 as the slice size, and, for the second and subsequent children, the previous
9 child's slice size is adjusted to be the minimum of its previous slice size and its
10 S2. The last child's slice size, however, is adjusted to be the minimum of its
11 previous slice size and the maximum of its D1 and D2, thus completing the first
12 iterative loop.

13 Using the parent's angle ω , the distance DP from the parent in the unit
14 disk can then be calculated as the greater of $\tanh(\text{asinh}(ST/2\omega)/2)$ or 0.5. DP is
15 saved as part of the data relevant to the parent node.

16 For each child, the second iterative loop begins by calculating the angle
17 $(S/ST)2\omega$, where S is the saved separation for the child. The angle $(S/ST)2\omega$ is
18 added to a running total which began at -2ω . The running total is saved with
19 other data relevant to the child.

20 Math routines 226 can then calculate a new angle for the child by calling
21 a function similar to the function "inside-angle" at columns 67 and 68 of U. S.

1 Patent 5,590,250, incorporated herein by reference. This function, referred to
2 herein as "InsideAngle", starts with a distance ("dist") that has been moved into a
3 wedge and an angle that is half of the wedge. InsideAngle takes as the
4 operative angle the smaller of the starting angle and $(\pi - \epsilon)$ where ϵ may have a
5 very small value such as 0.0001, thus avoiding problems in calculation of
6 arctangent. InsideAngle obtains the transformation that would move a point at
7 coordinates (dist, 0) on the unit circle to the origin. InsideAngle then applies this
8 transformation to the complex coordinates of a point at the intersection of the
9 perimeter of the unit circle with the ray starting at the origin whose angle with the
10 horizontal is the operative angle. InsideAngle returns as the resulting angle the
11 angle from the horizontal of the ray from the origin through the transformed
12 point.

13 To obtain a child's angle, InsideAngle is called with the distance DP and
14 with an angle calculated by multiplying the child's slice size from the first
15 iteration by $2\omega/ST$. The angle returned by InsideAngle is compared with $\pi/2$,
16 and the child's angle is the smaller of the two.

17 Before saving the child's new angle, math routines 226 save the child's
18 previous angle. If the absolute value of the difference between the old and new
19 angles exceeds a minimum value, math routines 226 also save data indicating
20 that layout should continue, as discussed below.

21 Finally, the second iterative loop obtains a child's area or side space by
22 calling a function similar to the function "room-available" at columns 67 and 68 of

1 U. S. Patent 5,590,250, incorporated herein by reference. This function, referred
2 to herein as "RoomAvailable", starts with a distance D that has been moved into
3 a wedge and an angle Φ that is half of the wedge. RoomAvailable returns a
4 distance to the edge of the wedge that is calculated by first obtaining the ratio
5 $(1-D^2)/2D$, and by then dividing the ratio by $\sin\Phi$ to obtain an initial distance S.
6 RoomsAvailable then returns the distance $((S^2-1)^{1/2}-S)$. To obtain a child's area,
7 RoomAvailable is called with the same distance and angle that were used in
8 calling InsideAngle, as described above. The distance returned by
9 RoomAvailable is saved as a measure of the child's area.

10 Although the software implementation described above can save
11 additional data, it is based on the discovery that only two items of data need to
12 be stored for each node in order to be able to perform layout and mapping as
13 described herein and in relation to Fig. 6 of copending coassigned U.S. Patent
14 Application 09/CCC,CCC (Attorney Docket No. D/98205Q2), entitled "Mapping a
15 Node-Link Structure to a Rendering Space Beginning from any Node",
16 incorporated herein by reference. One item indicates a distance or position
17 displacement from the node to its children nodes in the hyperbolic plane. The
18 other is an angle displacement in the hyperbolic plane between the extension of
19 the incoming link to the node's parent and outgoing link from the parent to the
20 node. These two items of data, or a handle that can be used to access them,
21 can be included in a link's data item in a directed graph data structure as
22 illustrated in Fig. 6 of copending coassigned U.S. Patent Application

1 09/EEE,EEE (Attorney Docket No. D/98205Q4), entitled "Node-Link Data
2 Defining a Graph and a Tree Within the Graph", incorporated herein by
3 reference.

4 The test in box 380 applies an appropriate criterion to determine whether
5 to continue layout to the next generation of nodes. As noted above in relation to
6 box 374, the criterion can be whether any child node's angle has been modified
7 by more than a small angular difference, such as 0.00001. If so, layout should
8 continue.

9 In box 382, walker routines 222 push the ID of each child node that is
10 expanded or not a leaf onto the front of the queue. Other child nodes could be
11 marked walked in box 382, since they do not have children that will be laid out.
12 When box 382 is completed or if the test in box 380 determines not to continue
13 or the test in box 370 determines that the node is already walked, the back node
14 on the queue is popped, in box 384, before returning to box 360.

15 Fig. 7 illustrates how layout of a changed node-link structure can be
16 performed in boxes 312, 324, and 334 in Fig. 5. In each case, layout begins in
17 response to a call that leads to layout and mapping, as shown in box 400. As
18 illustrated by the branch in box 402, however, the manner in which layout is
19 performed depends on the type of change being made in the node-link structure.

20 If the change is a change in orientation of the root node in response to an
21 orientation event, walker routines 222 can call math routines 226 to lay out the

1 root node at the new orientation before mapping and painting, in box 404. The
2 root node can be laid out as described above in relation to box 352 in Fig. 6, but
3 with the new orientation. The new orientation will then be used in mapping,
4 changing the orientation of the representation.

5 If the change is a non-animated edit, as could occur in response to a
6 stretch event, a drag event, a bookmark event, or a click event if edits are
7 pending, walker routines 222 first set up a list of remove edits, in box 410, and
8 then lay out the remove edits before mapping and painting, in box 412. Then,
9 walker routines 222 set up a list of add edits, in box 414, and then lay out the
10 add edits before mapping and painting, in box 416.

11 In the current implementation, the lists of edits are set up based on edit
12 source lists that are maintained by various routines in memory 214, including
13 grapher routines 220 and painter routines 224. Also, the current implementation
14 relates to a tree defined by expanded links, as explained in copending
15 coassigned U.S. Patent Application 09/EEE,EEE (Attorney Docket No.
16 D/98205Q4), entitled "Node-Link Data Defining a Graph and a Tree Within the
17 Graph", incorporated herein by reference. One pair of edit source lists,
18 designated "CollapsedLinks" and "ExpandedLinks" herein, includes edits for
19 links selected by contract requests and expand requests, respectively, and can
20 therefore be set up in box 330 in Fig 5. The other pair, designated
21 "RemovedLinks" and "AddedLinks" herein, includes edits for links that are

1 deleted and inserted, respectively. Multiple copies of the edit source lists may
2 exist for different purposes.

3 The list of remove edits set up in box 410 is based on RemovedLinks,
4 while the list of add edits set up in box 414 is based on AddedLinks. In setting
5 up a list in box 410 or 414, walker routines 222 access each edit in the
6 appropriate edit source list, and use the edit to obtain appropriate entries for the
7 list being set up. In each case, an edit in the edit source list is used to obtain a
8 node ID of the child node of the edit's link and an edit identifier indicating the
9 type of edit being performed.

10 A parent of the child node is added to the back of a list of affected nodes
11 unless it is already on the list. The parent is the child node's expanded parent
12 node or, if none of its parent nodes are currently expanded, its first parent node.
13 If an edit from an edit source list relates to a child node that does not have a
14 parent, it must relate to the root, and the root node is therefore put on the back
15 of the list of affected nodes in that case.

16 At the end of the iteration for a link from a source edit list, the link's child
17 node is also added to a list of children nodes before accessing the next edit in
18 the appropriate edit source list for use in the next iteration. In this way, an
19 iteration is performed for each edit on the edit source list until all edits have
20 been handled to complete the lists of affected nodes and children nodes.

1 Then, in box 412 or 416, the edits are laid out using the lists, following a
2 sequence similar to that in boxes 354 through 382 of Fig. 6 for each node in the
3 list of affected nodes, pushing the node from the list rather than the root node
4 onto the front of the queue and making several changes in box 372 as follows:
5 In addition to identifying which children are nonexpanded leaves, layout in boxes
6 412 and 416 determines whether each child is on the list of children nodes. If
7 so, layout multiplies the angle and radius for the child by a weight. In box 412,
8 the weight is zero, so that the child is laid out in box 374 at approximately its
9 previous position with an angle and radius of zero, and thus disappears. In box
10 416, the weight is one, so that the child is laid out in box 374 at its new position
11 with its full angle and radius.

12 The operations in boxes 410 through 416 could also be implemented
13 within an animation sequence, in which case remove edits could be handled
14 during an initial part of the animation sequence and add edits could be handled
15 during a subsequent part of the sequence. If on the other hand the non-
16 animated edits result primarily from node creation during painting, as described
17 above in relation to Fig. 5, the edits may only be add edits, and all currently
18 pending edits could be handled in each step of the animation sequence.

19 If the change is an animated edit, as could occur in response to an
20 insert/delete event such as a request to contract or expand an element, walker
21 routines 222 first obtain the numbers of elements to be removed and added
22 based on the source edit lists, in box 420. The number to be removed can be

1 obtained by adding the numbers of elements in CollapsedLinks and
2 RemovedLinks, while the number to be added can be obtained by adding the
3 numbers of elements in ExpandedLinks and AddedLinks. Then, in box 422,
4 walker routines 222 allocate the available animation steps between removing
5 steps and adding steps, and also set up lists of remove edits and add edits,
6 somewhat as in boxes 410 and 414 in Fig. 6. A simple allocation of animation
7 steps is half removing steps and half adding steps, but if there are no elements
8 to be removed, all the steps can be adding steps, and vice versa if no elements
9 to be added.

10 In setting up lists of remove edits and add edits in box 422, walker
11 routines 222 can perform as described above in relation to boxes 410 and 414
12 unless there are collapsed or expanded nodes. In the case of collapsed or
13 expanded nodes, the node itself, in addition to its parent, is pushed onto the
14 back of the list of affected nodes; then, the children of the node, rather than the
15 node itself, are added to the list of children nodes. In other words, collapsing or
16 expanding can be thought of as affecting two generations of nodes, unlike other
17 operations that only affect one. Walker routines 222 set up two pairs of lists of
18 affected and children nodes, one pair for remove edits and one for add edits.

19 The animation steps that remove nodes are then performed in the loop
20 that begins with box 430, with a weight being obtained, with the remove edits
21 being laid out with the weight, and with mapping and painting of an animation
22 frame, in box 432. Similarly, the animation steps that add nodes are then

1 performed in the loop that begins with box 440, with a weight being obtained,
2 with the add edits being laid out with the weight, and with mapping and painting
3 of an animation frame, in box 442. By removing nodes before adding nodes, the
4 situation is prevented where the same node would appear in two places in a
5 single frame. By performing a final step with the weight zero after removing
6 nodes and another with the weight one after adding nodes, the technique can
7 ensure that the final weight is zero or one, respectively.

8 A weight can be obtained in box 432 by subtracting the current remove
9 animation step number from the number of remove animation steps, then
10 dividing the difference by the number of remove animation steps, so that the
11 weights go from one to zero during a series of remove animation steps.
12 Similarly, a weight can be obtained in box 442 by adding one to the current add
13 animation step number, then dividing the sum by the number of add animation
14 steps, so that the weights go from approximately zero to one during a series of
15 add animation steps.

16 The total number of animation steps, coupled with the animation speed,
17 helps influence the perception of object constancy during animation. As can be
18 understood from the above description of how the weights are obtained, the total
19 number of animation steps determines the rate at which a removed or added
20 element's area changes, thus indirectly determining the rate at which other
21 elements must move in relation to the area of the removed or added element. A
22 larger number of animation steps, appropriately allocated between removing

1 steps and adding steps, is more likely to produce object constancy, provided a
2 sufficient animation speed is maintained.

3 The technique of Fig. 7, when performed with an appropriate number of
4 animation steps and at an appropriate speed, has successfully produced the
5 perception of a set of nodes contracting and expanding somewhat like a fan
6 would be folded and unfolded. By adjusting the radii and angles that are
7 assigned to the nodes, different perceptions can be obtained, such as that
8 deleted nodes are drawn into their parent or are squeezed off to infinity or that
9 inserted nodes grow out of their parent or are pulled in from infinity. When only
10 one of a group of children is deleted, it can appear to be squeezed off to infinity,
11 but if all of the children are deleted as a group, as in contraction, all can appear
12 to be drawn into their parent. Similarly, when one child is added to a group, it
13 can appear to be pulled in from infinity, but if all of the children are inserted as a
14 group, as in expansion, all can appear to grow out of their parent. Further,
15 grandchildren can be squeezed off to infinity while children are drawn into their
16 parent, with the rates adjusted so that the grandchildren appear stable and only
17 the children appear to move.

18 C.4. Variations

19 The implementation described above could be varied in many ways within
20 the scope of the invention.

1 An implementation similar to that described above has been successfully
2 executed on processors of IBM compatible PCs, but implementations could be
3 executed on other machines with any appropriate processors.

4 An implementation similar to that described above has been successfully
5 executed using C++ in 32-bit Windows environments, but other programming
6 languages and environments could be used, including non-object-oriented
7 environments, and other platforms could be used, such as Lisp, a Unix
8 environment, ANSI C, Pascal, and so forth.

9 An implementation similar to that described above has been successfully
10 executed with node-link data presented in an XML-compliant format and in an
11 experimental format, but the invention could be implemented with any suitable
12 type of node-link data, whether static or dynamic, and accessible in any
13 appropriate way, such as in memory or over a network.

14 An implementation similar to that described above has been implemented
15 with each iteration preparing and presenting one representation or an animated
16 series of representations of a graph in response to a navigation signal, but the
17 invention could be implemented with other types of iterations invoked by other
18 types of signals or calls.

19 An implementation similar to that described above has been successfully
20 executed with navigation signals received from a keyboard and mouse and
21 relating to a displayed representation or animated series of representations of a

1 node-link structure like the representations disclosed in Lamping et al., US-A-
2 5,619,632 and in copending coassigned U.S. Patent Application 09/AAA,AAA
3 (Attorney Docket No. D/98205), entitled "Presenting Node-Link Structures with
4 Modification", both incorporated herein by reference. The invention could,
5 however, be implemented with or without navigation signals; for example,
6 elements could be moved around in response to different sortings of the children
7 of a node or in response to the application of different filters to elements of a
8 structure. Also, the invention could be implemented with any appropriate type of
9 expand and contract signals or other navigation signals, including signals
10 resulting from external queries, selection of a menu entry-like item requesting
11 expansion below an indicated node or link, or selection of a menu entry-like item
12 requesting expansion below the current focus. The navigation signals could
13 instead relate to an illusory space like those produced by videogames or virtual
14 reality environments or a presentation space other than a display and navigation
15 signals could instead be produced by any appropriate user input device,
16 including other kinds of pointing devices and other kinds of devices for receiving
17 alphanumeric or linguistic input such as voice, gestures, or other modes of user
18 input. Further, the invention could be implemented with other types of
19 representations of node-link structures. The invention could be implemented
20 without animation or with any appropriate animation techniques.

21 The implementation described above obtains nearby relationship data
22 relating to relationships among a parent, its children, and its grandchildren, but

1 the invention could be implemented to obtain nearby relationship data relating to
2 a different set of nearby node-link relationships, such as relationships that
3 include more distant relatives of an element being laid out, whether additional
4 generations above or below the element or additional lateral relatives. For
5 example, grandparents of the element being laid out could be considered.

6 The implementation described above obtains layout data indicating, for a
7 node, a position displacement and an angle displacement from its parent. The
8 invention could, however, be implemented to obtain layout data for links, rather
9 than or in addition to nodes. Further, the invention could be implemented to
10 obtain layout data indicating relative position in any other appropriate way and
11 indicating additional information.

12 The implementation described above obtains counts of grandchildren of
13 an element's parent, and then uses the counts to obtain angles and radii of the
14 parent's children, which in turn are used to obtain a position displacement and
15 an angle displacement. The implementation performs specific computations as
16 described above. The invention could be implemented to obtain nearby
17 relationship data and layout data in any other appropriate way. For example,
18 each child of a parent could have a weighting and the relative weightings of the
19 children could be used to determine the space occupied by each child or to
20 determine proportional distances from the children to the parent. Also,
21 information about the available area around a node could be taken into account

1 by obtaining a layout with the least overlap in areas. Layout results could be
2 cached to minimize repetitive computation.

3 In the implementation described above, a node-link structure laid out in
4 the hyperbolic plane is then mapped into the unit disk and then painted in
5 accordance with copending coassigned U.S. Patent Application 09/CCC,CCC
6 (Attorney Docket No. D/98205Q2), entitled "Mapping a Node-Link Structure to a
7 Rendering Space Beginning from any Node", incorporated herein by reference,
8 but a node-link structure laid out in accordance with the invention could be laid
9 out in any other appropriate negatively curved space, and then be handled in
10 any other appropriate way, with or without mapping, or mapped and presented in
11 any other appropriate way, including mapping it into any other appropriate
12 rendering space and presenting it in any other appropriate display space,
13 including three-dimensional rendering and display spaces.

14 The implementation described above determines whether to lay out a
15 node's children by comparing an angle displacement obtained for the node with
16 a previous angle displacement, but the invention could be implemented by laying
17 out all descendants of each laid out node or by applying any other suitable
18 criterion to determine which elements to lay out.

19 The implementation described above is suitable for laying out elements of
20 a tree. The invention could be used to lay out elements of other types of node-
21 link structures, such as graphs in general.

1 The implementation described above uses node-link data that include
2 expansion flags of links to define a tree within a graph as disclosed in copending
3 coassigned U.S. Patent Application 09/EEE,EEE (Attorney Docket No.
4 D/98205Q4), entitled "Node-Link Data Defining a Graph and a Tree Within the
5 Graph", with memory management as disclosed in copending coassigned U.S.
6 Patent Application 09/DDD,DDD (Attorney Docket No. D/98205Q3), entitled
7 "Controlling Which Part of Data Defining a Node-Link Structure is in Memory",
8 both incorporated herein by reference, but the invention could be implemented
9 with a node-link structure defined in any other appropriate way, and loaded into
10 memory in any appropriate way.

11 The implementation described above employs a directed graph data
12 structure in which a link is represented as an item in two linked lists, one for the
13 outgoing links from its from-node and one for the incoming links to its to-node.
14 Any other suitable data structure could be employed.

15 The implementation described above can handle directed graphs,
16 including cyclic directed graphs, but the invention could be implemented for
17 other types of graphs by converting other types of links to appropriate
18 combinations of directed links or by otherwise providing a protocol for mapping
19 the structure of a graph to a tree. For example, an undirected link between two
20 nodes could be converted to a pair of directed links between the same nodes or
21 could be assigned a direction based on an appropriate criterion. In general, a
22 representation in which all undirected links have been converted to a pair of

1 directed links is likely to be visually confusing, because each pair of directed
2 links results in a cycle, but this confusing might be overcome by presenting
3 cycles in another way.

4 In the implementation described above, acts are performed in an order
5 that could in many cases be modified. For example, a depth-first walk rather
6 than a breadth-first walk could be performed in Fig. 6.

7 Also, in the implementation described above, several software portions
8 are distinguished, such as grapher, walker, painter, and math routines and the
9 client, but the invention could be implemented with other combinations of
10 hardware and software and with software organized in any appropriate way.

11 D. Applications

12 The invention has been applied in providing an interactive browser of
13 node-link structures. The invention could be applied in a variety of contexts in
14 which node-link structures are laid out for visualization. In particular, the
15 invention could be applied in visualizing web-related structures such as the
16 structure formed by a cached set of web pages or other web objects.

17 More generally, the invention could be applied to provide a browser for
18 organization charts, file system hierarchies, hypertext hierarchies, world wide
19 web connectivity structures, parts breakdowns, SGML structures, or any other
20 large node-link structures. The browser could be used in editing structures or
21 their contents.

1 E. Miscellaneous

2 The invention has been described in relation to software implementations,
3 but the invention might be implemented with specialized hardware.

4 Although the invention has been described in relation to various
5 implementations, together with modifications, variations, and extensions thereof,
6 other implementations, modifications, variations, and extensions are within the
7 scope of the invention. The invention is therefore not limited by the description
8 contained herein or by the drawings, but only by the claims.